Cloud Economics of a Sovereign OSS Stack: A Regulated Capital-Markets Migration Case Study

Zach Kelling — Hanzo Team

Version 1.0 2026

Abstract

We document the infrastructure economics of migrating a regulated capital-markets platform from a sprawling multi-project Google Cloud (GCP) footprint to a consolidated, self-hostable open-source stack built with Hanzo's agentic engineering system. The client—a FINRA/SEC-regulated securities platform, anonymized here—was spending, by *actual billing* once a startup-credit program expired, approximately $35,158/month (≈ $421,900/year) across eleven billing accounts, the great majority consumed by legacy GKE clusters and GPU virtual machines left over from an earlier architecture. Consolidation onto the new stack and owned hardware reduced spend to approximately $3,500/month (≈ $42,000/year)—a ∼90% reduction, on the order of $380,000/year saved—and removed the dependency on a single cloud vendor. The same agentic stack also *built* the platform: a full SEC-registered ATS with broker-dealer and transfer-agent functions, consolidated from 200+ microservices and 3,071,930 active lines of a prior implementation down to 1,056,566 active lines (≈70% less code; ∼3.5 M lines of dead code deleted) running as 3 compiled Go binaries plus a small set of OSS platform services, carried through FINRA/SEC approval and SOC 2. We decompose where the savings come from, report the figures with their provenance, and argue the same mechanism applies directly to any regulated enterprise or government modernization—defense among them— carrying a metered-API-plus-hyperscaler-sprawl cost structure.

Executive Summary. A regulated financial client appeared to be spending only a modest amount on Google Cloud because a startup-credit program was absorbing the bill. When those credits expired, the *actual* cost surfaced: about $35,158 a month—roughly $926 a day of newly cash-billed spend—almost all of it on old GKE clusters and GPU machines the new architecture had made unnecessary. Migrating to a consolidated, self-hostable open-source stack on owned hardware cut that bill to about $3,500 a month (≈90%, on the order of $380,000 a year), and removed the dependency on a single cloud vendor; ongoing consolidation targets a further floor near $1,200/month (nearly 97 percent off the original bill) as the last legacy projects are retired. Just as important is *how little code* now runs that platform: the agentic engineering stack collapsed 200+ legacy microservices into 3 compiled Go binaries plus a small set of OSS platform services, cutting the active codebase from ∼3.07 M to ∼1.06 M lines (deleting roughly 3.5 M lines of dead code and eliminating 1.3 GB of vendored dependencies)—and still cleared the bar of a heavily regulated, audited domain (FINRA/SEC approval, SOC 2). For any organization carrying a similar cost structure—a healthcare provider, a financial firm, a critical-infrastructure operator, or a government contractor—the lesson is concrete: the same approach modernizes an existing stack to a sovereign, self-hosted footprint that is far cheaper to run, smaller to maintain, and faster to build.

Contents

1 Before: A Sprawling, Credit-Masked Footprint

The starting state is typical of an organization that grew fast on Google Cloud under promotional credits. Headline spend looked small because a startup-credit program was absorbing it; an audit of the *actual* GCP billing console—after the credits expired—put true spend at approximately **$35,158/month** (≈$421,900/year), concentrated in legacy infrastructure rather than the live product. When the credits lapsed, roughly **$27,783/month** of previously credit-masked spend—about $926/day—flipped from a hidden line to cash. The footprint spanned **185 GCP projects** (27 billing-enabled, only ∼4 actually needed) across **11 billing accounts**:

Billing account	**$/month**	**$/year**
Legacy / startup-credit account (the waste)	27,783	333,396
Production account A	4,040	48,480
Secondary product group	3,087	37,044
New-stack non-prod (already efficient)	248	2,976
Total (actual billing)	**∼35,158**	**∼421,900**

Table 1: Actual monthly GCP spend by billing account, taken from the billing console after startup credits expired (11 billing accounts in total; the four material ones shown). A single legacy account accounted for ∼79% of the total; the current-generation stack was already a rounding error.

The legacy spend bought capacity the new design no longer needed: more than **125 GKE nodes across 20 clusters** (single clusters of 19, 17, and 14 nodes; 64 of those nodes were 18 GB machines), two standing **GPU virtual machines** (an A100-class instance at ∼$2,900/month and an L4-class instance at ∼$800/month, ∼$3,700/month together), **5 Cloud SQL PostgreSQL instances**, and **∼18.5 TB** of persistent disk—several terminated VMs were even still paying for idle attached storage. Two problems compound here: **cost** (most of the spend bought unused capacity) and **sovereignty** (the entire footprint lived inside one hyperscaler—GKE, Cloud SQL, BigQuery, GCS, Artifact Registry, Cloud Build—with the data gravity and lock-in that implies).

2 The Migration: Two Documented Steps

The migration consolidated the workload onto Hanzo's open-source stack—the same self-hostable components (identity, key management, data, gateway, agent runtime) that run sovereign and disconnected—and onto owned hardware for the compute-heavy services. The savings landed in two measured steps:

Decommissioning was clean precisely because the legacy capacity was unused: shutting down the two GPU VMs removed ∼$3,700/month on its own; the largest legacy GKE clusters and dozens of dead projects accounted for most of the rest. The net effect is a reduction from ∼$35,158 to ∼$3,500 per month (≈90%) and, equally important, the elimination of single-vendor lock-in: the resulting stack runs on owned hardware, in another cloud, or air-gapped.

3 Where the Savings Come From

The reduction is not a one-time clean-up; it is structural, and it reproduces:

- **No metered model markup.** Owned models (the Zen family) replace per-token API calls, removing a recurring per-inference charge.

State	$/month	vs. start
Actual billing (credits expired)	∼35,158	—
Step 1 — retire legacy: stop 2 GPU VMs (−$3,700), delete legacy GKE clusters, close dead projects	∼6,489	−82%
Step 2 — consolidate & right-size onto the owned stack and owned hardware	∼3,500	−90%
Target — finish consolidation (retire last legacy projects)	∼1,200	≈−97 pts

Table 2: Reduction to date. Step 1 (retiring unused legacy) alone removed ∼82% of spend with no impact on the live product; Step 2 consolidated the remaining workload onto the self-hostable stack, landing at ∼$3,500/month (≈$380,000/year saved). The ∼$1,200/month line is a forward *target* as the last legacy projects are retired—not yet realized.

- **No hyperscaler margin.** Self-hosting on owned hardware removes the cloud provider's margin on compute, storage, and egress.
- **Efficient transport.** The ZAP protocol's 91% memory and 96% energy reductions and 30× throughput mean the same work needs far less hardware (see the ZAP paper).
- **Consolidation.** One coherent stack replaces a sprawl of projects, clusters, and services—the single largest line item above was pure redundant overhead.

Together these are the mechanism behind the roughly order-of-magnitude infrastructure savings, and they apply to any organization carrying a metered-API plus hyperscaler-sprawl cost structure.

4 The Build: Consolidation, Not Volume

The economics of *running* the platform are only half the story; the economics of *building* it are the other half—and here the measured story is *doing more with dramatically less code*. The legacy estate was a sprawl of **200+ microservices** (a single legacy auth monolith alone fronted 391 route handlers; ∼80+ distinct deployable services, 49 of them active Node.js processes at the time of migration). The same agentic engineering stack that now runs the platform also built it, and the artifacts are small: **3 compiled Go binaries** (ATS / broker-dealer / transfer-agent) alongside ∼10 OSS platform services and 5 validator nodes. Measured git-over-git, the active codebase fell from **3,071,930** to **1,056,566** lines (≈70% less; roughly 3.5 M lines of dead code deleted):

This inverts the usual "big system = big codebase" intuition: agent-driven engineering produced a *leaner* system than the one it replaced, which is why it is cheaper to run (fewer moving parts), cheaper to maintain (less code), and faster to deploy (<2 min rolling deploys, versus ∼1 hr manual or 15–30 min canary before). The memory story is the same shape: the legacy fleet held ∼1 TB+ of *provisioned* cluster RAM (64 nodes × 18 GB) against an application working set of roughly 10 GB; the consolidated Go binary holds its working set in ∼50 MB (≈99.5% less), and cold start dropped from ∼10 s per Node.js service to <1 s. On security, an internal source audit found **93 findings** across 127+ repositories (22 critical, 30 high, 32 medium, 9 low) plus ∼780+ npm dependency vulnerabilities; the consolidated rebuild remediated that entire class to *zero actionable* at the architecture level (header hardening, strict CORS allowlisting, JWKS-verified JWTs, secrets moved out of plaintext environment, and the 1.3 GB vendored dependency tree—the source of most of the npm vulns—eliminated outright).

Measure	Legacy	New stack	Change
Deployable services	200+	3 Go binaries	∼60× fewer
Active lines of code (whole estate)	3,071,930	1,056,566	≈70% less
Lines of code (trading/auth core)	∼263,803	∼44,308	∼6× less
Vendored dependencies (flagship svc)	1.3 GB	0	eliminated
Provisioned cluster RAM	∼1 TB+	∼50 MB working set	∼99.5% less
Deploy artifact (flagship)	—	48 MB binary	replaces 17+ services
Deploy time	∼1 hr manual	<2 min rolling	—
Cold start (per service)	∼10 s	<1 s	∼10× faster
Automated tests (new stack)	n/a tracked	700+ Go tests	—

Table 3: Measured engineering footprint, new stack vs. the legacy implementation it replaced, git-verified. The whole-estate line count fell from ∼3.07 M to ∼1.06 M active lines; the trading/auth core specifically went ∼6× smaller. A single 48 MB Go binary subsumes 17+ formerly separate services.

5 The Compliance Bar

The load-bearing proof point is that this lean, cheap-to-run system is not a prototype: it is a regulated securities venue—an SEC-registered alternative trading system with broker-dealer and transfer-agent functions—carried through **FINRA/SEC approval** and **SOC 2** compliance, in record time, with a **small team**. Agent-driven engineering shipped and passed regulatory scrutiny in financial services, among the most compliance-heavy software environments that exists.

6 Implications for Regulated and Government Modernization

The same motion transfers directly to any regulated-enterprise or government modernization—a healthcare system, a bank, a utility, or a government contractor (defense included). An organization with an existing, audited (and, for government systems, already-accredited) software stack does not need to rebuild from zero; the agentic stack can modernize that stack in place—to a post-quantum, sovereign, self-hostable footprint—while the consolidation cuts recurring infrastructure cost by an order of magnitude, shrinks the codebase that must be maintained (and, where applicable, accredited), and removes single-vendor dependency. The wedge is low-friction (modernize what you already have, with proofs), the savings are immediate and structural, and the result is software the operator owns and can run where the data lives. A regulated financial platform is the existence proof; the same pattern applies anywhere the cost structure and the compliance burden repeat.

A note on figures. Cost figures are drawn from the client's actual GCP billing-console data after promotional credits expired (a more defensible basis than list-price estimates); infrastructure, code, and security figures are drawn from internal infrastructure, code, and source-security audits. Line counts, service counts, and dependency sizes are measured (git-verified) where stated. Where a number is a forward target rather than a realized result (e.g. the ∼$1,200/ month consolidation floor) it is labeled as such; where a source labels a number projected or self-reported, we have not relied on it here. All identifiers are removed; the client is not named.